Subject: SMCIT03EBenowitzAbs.doc
From: "Amalaye Oyake" <amalaye.oyake@jpl.nasa.gov>
Date: Mon, 9 Dec 2002 17:15:04 -0800
To: "Eddie Benowitz" <Edward.G.Benowitz-117142@jpl.nasa.gov>

Java for Flight Software

Author: Edward G. Benowitz and Albert F. Niessner
Affiliation: NASA Jet Propulsion Laboratory
Address: 4800 Oak Grove Drive
Pasadena, California 91109
Email:eddieb@mail2.jpl.nasa.gov, Albert.F.Niessner@jpl.nasa.gov
Phone: 818-393-7169

Preferred Type: Oral Presentation
Abstract:
This work involves developing representative mission-critical spacecraft software using the Real-Time Specification for Java(RTSJ).

This work currently leverages actual flight software used in the design of actual flight software in the NASA's Deep Space 1 (DS1), which flew in 1998.

The flight software is completely rewritten in Java code and is based on the original flight code.

The software has been redesigned completely by utilizing the best practices in OO design. Thus we have successfully re-architected the code, as opposed to performing a line-by-line port, we are the code, using.

With this new implementation, we were able to successfully demonstrate a portion of the spacecraft attitude control and fault protection subsystem, running on a standard Java platform. Our goal is to run on flight-like hardware, in closed-loop with the original spacecraft dynamics simulation. The advantages of the features provided by the Real Time Specification for Java (RTSJ), will be very helpful towards accomplishing our objective.

In re-designing the software from the original C code, we have made a number of observations on adopting OO techniques for flight software development. First, we have taken advantage of design patterns, and have seen a strong mapping between certain patterns and flight software approaches. We gain advantages from the state design pattern, eliminating the need for long, error-prone switch statements. The facade pattern is used for communication between threads, hiding queues where necessary, or allowing direct calls. To ensure the correctness of measurement units, numerical computations are performed via compile-time checked measurement unit classes. This technique allows earlier bug detection in numerical code. As an additional observation, we directly map elements of the control loop to objects.

Our approach places an emphasis on pluggable technology. Interfaces, in conjunction with a façade pattern, allow us to exposes only the behavior of a subsystem, rather than exposing its implementation details. Since the RTSJ reference implementation does not currently support debugging, we chose to apply pluggable technology to our scheduler and memory allocation interfaces.

Thus, we allow real-time client code to run on a standard Java VM, allowing the code to be debugged in a graphical development environment on any desktop PC at the cost of decreased real-time performance. Once non-real-time issues have been debugged, the real-time aspects can be debugged in isolation on an RTSJ-compliant virtual machine.

THESE ARE STRONG POINTS, but this paragraph needs strengtheningTechnically this looks like an important issue – you may want to say why there has been a need for new scheduling models and the relevance to FSW****What are the benefits of RTSJ???***As opposed to good old C????***With the addition of new scheduling and memory management features, come new failure modes and programming pitfalls. The developer must consciously avoid violating memory area rules, and must ensure that no memory leaks occur. We propose a series of guidelines for using the RTSJ memory management features. We provide a set of recommendations for memory allocation, showing scenarios that take advantage of memory areas provided by RTSJ. In addition, we place restrictions on memory allocation scenarios that are particularly error-prone.

Java for Flight Software

Author: Edward G. Benowitz and Albert F. Niessner
Affiliation: NASA Jet Propulsion Laboratory
Address: 4800 Oak Grove Drive
Pasadena, California 91109
Email:eddieb@mail2.jpl.nasa.gov, Albert.F.Niessner@jpl.nasa.gov
Phone: 818-393-7169

Preferred Type: Oral Presentation

Abstract:

This work involves developing representative mission-critical spacecraft software using the Real-Time Specification for Java(RTSJ). Utilizing a real mission design, this work leverages the original flight code from NASA's Deep Space 1(DS1), which flew in 1998. However, instead of performing a line-by-line port, the code is re-architected in pure Java, using best practices in OO design. We have successfully demonstrated a portion of the spacecraft attitude control and fault protection, running on a standard Java platform, and are currently in the process of taking advantage of the features provided by the RTSJ. Our goal is to run on flight-like hardware, in closed-loop with the original spacecraft dynamics simulation.

In re-designing the software from the original C code, we have made a number of observations on adopting OO techniques for flight software development, and we explain the benefits of this approach. We have taken advantage of design patterns, and have seen a strong mapping from certain patterns to the flight software. The state design pattern eliminates the need for long, error-prone switch statements. The facade pattern is used for communication between threads, hiding queues where necessary, or allowing direct method calls. To ensure the correctness of measurement units, numerical computations are performed via measurement unit classes that are checked at compile time. This technique allows earlier bug detection in numerical code. As an additional observation, we directly map elements of the control loop to objects.

Our approach places an emphasis on pluggable technology. Interfaces, in conjunction with a façade pattern, expose only the behavior of a subsystem, rather than exposing its implementation details. Since the RTSJ reference implementation does not currently support debugging, we chose to apply pluggable technology to the scheduler and memory allocation interfaces. Thus, real-time client code can be run on a standard Java virtual machine, allowing the code to be debugged in a graphical development environment on a desktop PC at the cost of decreased real-time performance. Once non-real-time issues have been debugged, the real-time aspects can be debugged in isolation on an RTSJ-compliant virtual machine.

With the addition of the RTSJ's scheduling and memory management features, come new failure modes and programming pitfalls. The developer must consciously avoid violating memory area rules, and must ensure that no memory leaks occur. This paper presents a series of guidelines for using the RTSJ memory management features. We provide a set of recommendations for memory allocation, showing scenarios that take advantage of memory areas provided by RTSJ. In addition, restrictions are placed on memory allocation scenarios that are particularly error-prone.